UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII - PATOS
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE GRADUAÇÃO EM BACHARELADO EM COMPUTAÇÃO

LAURA RAFAELLA FEITOSA ALVES DE OLIVEIRA

UM PROJETO DE INTERFACES COM O USUÁRIO BASEADO NA ABORDAGEM MICRO-FRONTENDS PARA O HYPERDRIVE

PATOS - PB
2024

LAURA RAFAELLA FEITOSA ALVES DE OLIVEIRA

UM PROJETO DE INTERFACES COM O USUÁRIO BASEADO NA ABORDAGEM MICRO-FRONTENDS PARA O HYPERDRIVE

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Computação do Centro de Ciências Exatas e Sociais Aplicadas da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Computação.

Orientador: Dr. Janderson Jason Barbosa Aguiar
Coorientador: Dr. Demetrio Gomes Mestre

PATOS - PB
2024

O48p Oliveira, Laura Rafaella Feitosa Alves de.
Um projeto de interfaces com o usuário baseado na abordagem micro-frontends para o hyperdrive [manuscrito] / Laura Rafaella Feitosa Alves de Oliveira. - 2024.
40 p. : il. colorido.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2024.
"Orientação : Prof. Dr. Janderson Jason Barbosa Aguiar, Coordenação do Curso de Computação - CCEA. "
"Coorientação: Prof. Dr. Demetrio Gomes Mestre , Coordenação do Curso de Computação - CCEA. "

1. Arquitetura de software. 2. Desenvolvimento Web. 3. Interface de programação. I. Título

21. ed. CDD 005.1

Elaborada por Camile de A. Gomes - CRB - 15/559 COBIB/UEPB

LAURA RAFAELLA FEITOSA ALVES DE OLIVEIRA

UM PROJETO DE INTERFACES COM O USUÁRIO BASEADO NA ABORDAGEM MICRO-FRONTENDS PARA O HYPERDRIVE

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Estadual da Paraíba — Campus VII, em cumprimento à exigência para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 18/06/2024

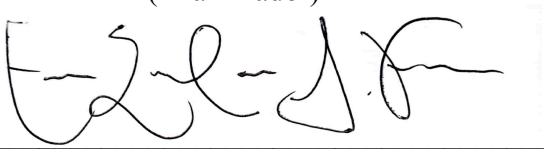
BANCA EXAMINADORA

Janderson Jason Barbosa Aguiar

Prof. Dr. Janderson Jason Barbosa Aguiar
(Orientador)

Harllem A. Nascimento

Prof. Bel. Harllem Alves do Nascimento
(Examinador)



Prof. Bel. Esdras Samuel de Araújo Ferreira
(Examinador)

Dedico a minha família, amigos, professores que contribuiram e todos que me apoiaram neste ciclo. Este trabalho só foi possível graças a vocês.

AGRADECIMENTOS

A Deus, por estar sempre comigo e por nunca me abandonar. Sua presença constante foi fundamental para superar os desafios e alcançar este objetivo.

Aos meus pais e família, pela dedicação e amor incondicional. Vocês são minha base e minha força para seguir em frente.

Ao meu gatinho Salém, que esteve comigo nesse período dedicado ao tcc.

Aos meus professores, orientador e coorientador, pelo conhecimento transmitido e pela orientação ao longo deste trabalho. Foram fundamentais para o meu crescimento acadêmico e profissional.

Aos meus amigos e colegas de classe, que estiveram comigo durante esta caminhada, vocês foram essenciais para que este momento acontecesse.

A todos que compartilharam comigo momentos de aprendizado, risos e desafios, agradeço por fazerem parte desta trajetória.

“A tecnologia é o campo de batalha onde o futuro é decidido."

— Tim Berners-Lee

RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação web baseada em micro-frontends (MFEs), com foco na interação com uma API fictícia que simula serviços na *blockchain*. Inicialmente, é realizada uma análise dos conceitos de MFEs. Em seguida, são elicitados os requisitos para a aplicação, seguido pelo desenvolvimento de um *frontend*, que interage com a referida API. Após a autenticação bem-sucedida, cada usuário recebe um token de acesso para navegar e realizar requisições relacionadas a projetos científicos. Observou-se que os MFEs na aplicação web destacou benefícios em modularidade e escalabilidade, estabelecendo uma base sólida para futuras expansões do sistema. Tecnologias como *Single-SPA, Node.js, Angular, React e Webpack* foram utilizadas durante o desenvolvimento. Como sugestão para trabalhos futuros, propõe-se realizar testes de usabilidade com usuários finais para avaliar a intuitividade e a eficiência da interface, bem como integrar a aplicação com uma API real para validar completamente.

Palavras-chave: Arquitetura de *Software*; Desenvolvimento Web; Micro-frontends.

ABSTRACT

This paper presents the development of a web application based on micro-frontends (MFEs), focusing on interaction with a fictitious API that simulates blockchain services. Initially, an analysis of MFE concepts is conducted. Subsequently, requirements for the application are elicited, followed by the development of a frontend that interacts with the aforementioned API. Upon successful authentication, each user receives an access token to navigate and make requests related to scientific projects. The use of MFEs in the web application demonstrated benefits in modularity and scalability, establishing a solid foundation for future system expansions. Technologies such as Single-SPA, Node.js, Angular, React, and Webpack were employed during development. As a suggestion for future work, usability tests with end-users are proposed to evaluate the intuitiveness and efficiency of the interface, along with integrating the application with a real API for comprehensive validation.

Keywords: Software Architecture; Web Development; Micro-frontends.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ARK	*Archival Resource Key*
CSS	*Cascading Style Sheets*
DARK	*Decentralized Archival Resource Key*
DOI	*Digital Object Identifier*
DSR	*Design Science Research*
HTML	*HyperText Markup Language*
MFE	*Micro-Frontend*
MVC	*Model View Controller*
ORCID	*Original Researcher and Contributer ID*
PID	*Persistent Identifier*
REST	*Representational State Transfer*
SPA	*Single-Page Application*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento *frontend* da web, também conhecido como desenvolvimento do lado do cliente, é a prática de produzir HTML, CSS e JavaScript para um site ou aplicação da web, de modo que o usuário possa visualizá-los e interagir diretamente com eles (Lindley, 2017). TOTVS (2021) cita que o *frontend* está muito relacionado com a interface gráfica do projeto. Em outras palavras, é onde ocorre o desenvolvimento da aplicação pela qual o usuário irá interagir diretamente, seja em programas de computador, sites, aplicativos e assim por diante. Por isso, é fundamental que o desenvolvedor se preocupe com a experiência do usuário.

Por um lado, existe a arquitetura monolítica, que se refere ao modelo de desenvolvimento de uma aplicação dentro de uma única estrutura executável. Essa abordagem é geralmente adotada para gerenciamento de projetos pequenos e simples. Por outro lado, existe os micro-frontends (MFEs), que são autossuficientes e independentes de implantação. Isso permite que equipes trabalhem simultaneamente no desenvolvimento de diferentes partes da aplicação, colaborando na mesma base de código ou projeto sem comprometer o trabalho um do outro.

Jackson (2019) destaca que no cenário atual de desenvolvimento a necessidade da arquitetura MFEs é essencial. Empresas enfrentam desafios ao lidar com códigos monolíticos no *frontend*, e a demanda por aplicações web inovadoras é crescente e recomendada por organizações como o *Thoughtworks Technology Radar*, como uma técnica a ser adotada quando apropriada. No entanto, à medida que os sistemas *frontend* e o cenário da web evoluem, surge uma necessidade premente de abordar as complexidades do desenvolvimento web moderno, particularmente quando se trata de interfaces que consomem serviços e microsserviços na internet.

O principal foco deste trabalho reside na resolução de um desafio primordial na concepção de um projeto de interfaces de usuário destinadas a simplificar o acesso à API REST *Hyper Drive*. O desafio se deu pela ausencia de uma camada *frontend* para acessar o *backend*. Esse requisito foi o ponto chave para impulsionar a busca pela melhor arquitetura, capaz de viabilizar essa integração de maneira eficaz e intuitiva.

A necessidade de obter uma solução que facilite o acesso a essa API surge mediante a tornar a interação com o serviço mais acessível e eficiente para os usuários finais, com o intuito de estudar, analisar e implementar a arquitetura mais adequada para garantir uma experiência satisfatória e fluida. Este motivo se difundiu pela imperatividade de resolver um problema específico e essencial no contexto da interface do usuário, tornando-se uma pesquisa relevante sobre a arquitetura mais adequada para alcançar esse propósito.

Este trabalho utiliza a abordagem MFEs, baseada na arquitetura de microsserviços, para a criação de interfaces que facilita a comunicação com o serviço de acesso à *blockchain*. Ela requer a criação de um projeto de interface com usuário eficiente e eficaz com os métodos disponibilizados por uma API específica, que é essencial para acessar uma rede *blockchain*. A integração dessa tecnologia ocorre por meio da API REST *Hyper Drive* que está sendo desenvolvida em *Flask*, e é responsável por conectar a interface de usuário *(frontend)*. Essa

integração é fundamentada no *"Ark Repository"*, que atua como um repositório de *blockchain* para armazenar a bibliografia e o acervo das bibliotecas das instituições de ensino superior do país, otimizando a experiência do usuário. Bass, Clements e Kazman (2012) ressalta que ao se desenvolver sistemas de *software*, a estrutura desses sistemas desempenha um papel fundamental na garantia de que as organizações atinjam seus objetivos.

O *frontend* desempenha um papel crucial nessa acessibilidade, pois é a camada que traduz a complexidade da *blockchain*, permitindo que os usuários interajam com a tecnologia de forma amigável. Para que esse acesso ocorra de maneira corrente e de alto desempenho, é essencial desenvolver interfaces que simplifiquem a interação do usuário com a API, tornando-a acessível, intuitiva e eficiente.

1.1 Objetivos

1.1.1 Objetivo Geral

O sistema Web a ser desenvolvido neste trabalho foi motivado pela criação de interfaces para a integração com *blockchain*, seguindo uma arquitetura de MFEs. Por este motivo, este trabalho pretende estudar soluções e as melhores práticas disponíveis no mercado para este desafio e, em seguida, desenvolver interfaces que possam gerenciar essa integração de forma eficaz. Para isso, foi utilizado a API REST *Hyper Drive* como o meio de ligação entre a *blockchain* e o *frontend*. O objetivo geral do trabalho consiste em desenvolver interfaces de *frontend* que permitirão aos usuários acessar e interagir com os dados da *blockchain* por meio da API.

1.1.2 Objetivos Específicos

Para se alcançar o objetivo geral deste trabalho, será necessário atingir os seguintes objetivos específicos:

- Realizar estudos de pesquisas bibliográficas sobre os conceitos e soluções existentes para uma arquitetura de *software* baseada em MFEs;
- Levantar os requisitos para criação das interfaces;
- Desenvolver uma aplicação *frontend* baseada nos conceitos de arquitetura de MFEs;

1.2 Resumo da metodologia

Na investigação realizada, foi conduzida uma pesquisa primária, que, conforme Wazlawick (2021), vai além de simplesmente realizar uma revisão bibliográfica. A pesquisa busca soluções por meio do desenvolvimento de interfaces para resolver o problema em questão. Esta pesquisa tem objetivos descritivos e exploratórios, utilizando um procedimento técnico que se aproxima do *Design Science Research*, segundo Dresch, Lacerda e Junior (2015).

Foram analisados trabalhos com diferentes abordagens arquiteturais para a implementação das interfaces. Após essa análise, foi escolhida a arquitetura mais adequada para o problema identificado, houve a elicitação dos requisitos necessários para o desenvolvimento e, por fim, procedeu-se ao desenvolvimento das interfaces.

1.3 Justificativa

Jackson (2019) descreve que na edição de novembro de 2016 do radar tecnológico da Thoughtworks, os MFEs foram inicialmente listados como uma técnica a ser avaliada. Posteriormente, foram promovidos para teste e, finalmente, recomendados para implementação, indicando que são vistos como uma abordagem comprovada que deve ser utilizada quando apropriado. Garantir que foi feita a escolha correta de arquitetura de *software* e padrões de projeto para a sua aplicação web *frontend* é algo que pode salvar muito tempo para o time de desenvolvedores e evitar muitas dores de cabeça no futuro (Filho, 2021).

Este trabalho visa desenvolver um sistema utilizando a arquitetura de MFEs. Ela permite expansão futura e oferece flexibilidade ao possibilitar o uso de diferentes tecnologias conforme a necessidade de cada módulo da aplicação. Essa liberdade de escolha não apenas impulsiona a inovação, mas também garante uma adaptabilidade dinâmica, essencial em um cenário de constante mudança tecnológica. Isso permite a escolha das melhores ferramentas para cada caso específico.

Além disso, essa arquitetura isola as funcionalidades, reduzindo o impacto de problemas em uma parte do sistema sobre as outras, facilitando a identificação e correção de erros, bem como a implementação de novas funcionalidades sem comprometer a estabilidade do sistema. Embora seja relativamente nova, a arquitetura de MFEs traz um diferencial significativo para o desenvolvimento e uso de aplicações web, oferecendo vantagens em termos de manutenção e escalabilidade.

1.4 Estrutura do trabalho

Este trabalho apresenta cinco capítulos e está organizado da seguinte maneira: no Capítulo 1, é apresentada uma visão geral deste trabalho em relação à contextualização do problema, objetivos, justificativa e estrutura do trabalho; no Capítulo 2, são apresentados os conceitos e trabalhos relacionados a esse trabalho de conclusão de curso; no Capítulo 3, são apresentados os aspectos metodológicos e desafios da arquitetura de MFEs; no Capítulo 4, resultados e discussões; no Capítulo 5, as considerações finais; e ao final, encontram-se as referências.

2 REFERENCIAL TEÓRICO

Neste capítulo estão presentes os tópicos necessários para o entendimento deste trabalho.

2.1 Conceitos Básicos

Uma técnica bem conhecida para identificar e localizar objetos na internet é a chamada identificação persistente (Hakala, 2010). Os identificadores persistentes, ou simplesmente PIDs *(Persistent Identifiers)*, são informações que representam única e permanentemente objetos/entidades de diferentes sistemas e contextos. Atualmente, os PIDs são utilizados de forma pervasiva para identificar unicamente artigos, livros, autores, documentos, arquivos, objetos de arte, páginas web, entre outros artefatos digitais na internet (Golodoniuc; Car; Klump, 2017). Na literatura, há diversas abordagens para geração de PIDs (McMurry et al., 2017), e uma delas, o *Archival Resource Key* (ARK), tem se destacado ultimamente devido ao aumento de popularidade de seu uso. Um dos principais motivos para essa popularidade é seu processo inovador e autossuficiente de atribuição de PIDs. Isto quer dizer que, durante o processo de geração de um novo PID, não há obrigatoriedade de que uma autoridade central reguladora intermedeie o processo, que é uma característica comum à maioria das abordagens do estado da arte.

No entanto, é importante ressaltar que a configuração/implementação de um gerador de ARK feita de forma inadequada, ou isolada, pode gerar riscos de perda de dados e duplicação de informação, inclusive permitindo que ARKs diferentes sejam gerados para o mesmo objeto em diferentes instituições. Quando problemas destas naturezas ocorrem, por vezes, é necessário contratar serviços especializados, que custam caro financeiramente. Considerando este cenário de riscos, havia uma demanda evidente por novas abordagens de geração de ARKs que promovessem transparência, fossem fáceis de usar, auditáveis e de baixo custo. Para endereçar essa necessidade, o *Decentralized Archival Resource Key* (dARK) foi proposto pelo Instituto Brasileiro de Inovação em Ciência e Tecnologia (IBICT) para descentralizar a atribuição de identificadores ARK baseada na tecnologia *blockchain* (Segundo et al., 2023). A ideia é atender da melhor forma possível as demandas de identificação de artefatos digitais em um mundo em constante transformação e inclusão digital.

O conceito-chave do dARK é empregar uma abordagem que viabilize a manipulação (colaborativamente), por diversas instituições, de seu sistema de geração de PIDs, para poderem atribuir e reutilizar identificadores persistentes sem a dependência de uma terceira parte (uma autoridade central reguladora) (Sicilia et al., 2019). Para tanto, o dARK utiliza uma infraestrutura baseada em *blockchain* e implementa o conceito de consórcios institucionais. Assim, o dARK é capaz de atender a uma série de requisitos, tais como: tolerar falhas do sistema PID; guardar a informação acerca do PID gerado permanentemente; analisar a procedência dos dados; promover interoperabilidade dos sistemas de atribuição de PID; gerar economia e democratização das informações.

Tais requisitos só podem ser atendidos devido à utilização da tecnologia *blockchain*, pois esta tecnologia promove o agrupamento de um conjunto de informações capazes de se conectar por meio de criptografia, viabilizando assim, a privacidade e segurança das informações armazenadas (que é segredo da não necessidade de uma autoridade central reguladora para intermediar o processo de geração de PIDs)(Bellini, 2019). Porém, com o benefício do uso da tecnologia *blockchain*, vem suas dificuldades. A utilização da tecnologia *blockchain* promove privacidade e segurança, no entanto, introduz uma perda de desempenho por promover um aumento no tempo necessário para persistir informações na rede *blockchain*. Além disso, o uso da rede *blockchain* não permite a integração do sistema de PID com sistemas legados existentes nas instituições que precisam gerar PIDs.

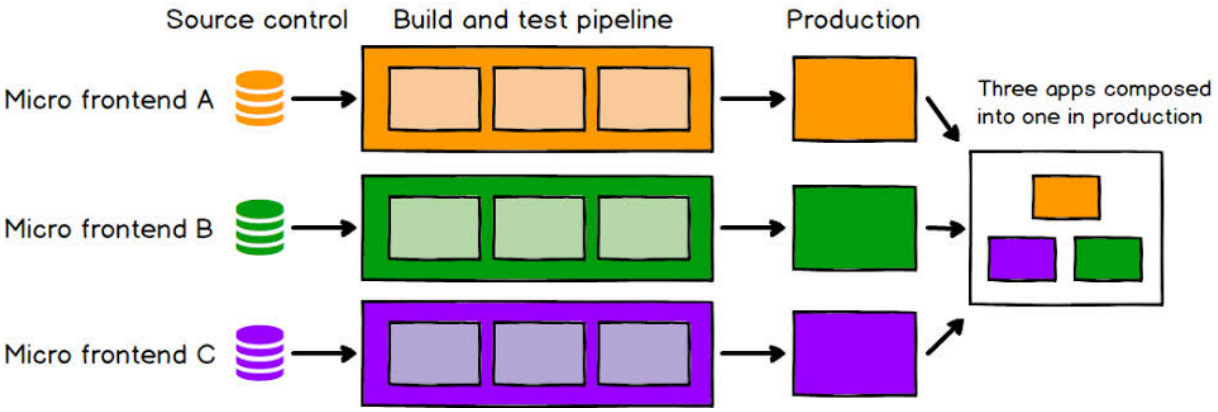
Assim, a principal ênfase deste trabalho reside na resolução de um desafio primordial na concepção de um projeto de interfaces de usuário destinadas a simplificar o acesso à API REST *Hyper Drive*, o serviço que acessa uma instância de dARK. Devido a ausência de uma camada de *frontend* para acessar à API REST *Hyper Drive*, esse requisito foi o ponto crucial para impulsionar a busca pela melhor arquitetura capaz de viabilizar essa integração de maneira eficaz e intuitiva.

2.2 Tecnologias de Arquitetura de Sistemas

O termo MFEs surgiu pela primeira vez no *ThoughtWorks Technology Radar* no final de 2016. Ele estende os conceitos de microsserviços para o mundo *frontend*. A tendência atual é construir um aplicativo de navegador poderoso e rico em recursos, também conhecido como aplicativo de página única, que se baseia em uma arquitetura de microsserviços. A ideia por trás do MFEs é pensar em um site ou aplicativo web como uma composição de recursos que pertencem a equipes independentes (GEERS, 2017). Jackson (2019) ressalta que os MFEs visam simplificar tarefas complexas ao dividir grandes projetos em partes menores e mais simples de gerenciamento. É fundamental identificar e detalhar claramente as conexões entre essas partes. As decisões em termos de tecnologia, código-fonte e métodos de lançamento devem permitir que cada componente opere e se desenvolva de forma independente, sem exigir uma cooperação excessiva entre eles.

Na Figura 1 é mostrado como funciona a arquitetura MFEs. Cada MFE é trabalhado de forma independente, utilizando suas próprias ferramentas e metodologias específicas durante o processo de desenvolvimento. Após a construção de cada um, esses componentes são integrados para compor um único produto final.

Figura 1 – Arquitetura MFEs



Fonte: Jackson (2019).

Flanagan (2012) ressalta que o JavaScript é uma linguagem de programação comumente utilizada na web, sendo essencial para a grande maioria dos sites modernos. Todos os navegadores atuais, incluindo os de computadores, dispositivos móveis e consoles de jogos, possuem interpretadores JavaScript integrados, o que torna essa linguagem amplamente utilizada. É uma das três principais tecnologias que qualquer desenvolvedor web deve dominar, juntamente com HTML e CSS, as quais são essenciais para especificar o conteúdo, apresentação e comportamento de páginas na internet.

HTML *(HyperText Markup Language)* é a principal linguagem utilizada na web. Ela permite a criação de documentos estruturados em títulos, parágrafos, listas, links, tabelas, formulários e em muitos outros elementos nos quais podem ser incorporadas imagens e objetos com, por exemplo, uma animação ou um video (Flatschart, 2011). O HTML será utilizado no site como uma ferramenta simples de marcação que, ajudará e irá aprimorar o CSS, ajudando no *frontend* do site e na organização (VELOSO et al., 2022).

TOTVS (2020) descreve que o *Cascading Style Sheet* (CSS), é empregado para estilizar elementos criados em linguagens de marcação em um site. Que desempenha a função de separar o conteúdo do site de sua representação visual, permitindo ajustar características como cores de texto, fontes e espaçamento entre blocos. Ele é responsável por controlar todo o aspecto estético de uma página da web, garantindo uma apresentação visual atraente e coesa.

O *React* é uma biblioteca *frontend* e tem como um de seus objetivos facilitar a conexão entre diferentes partes de uma página, portanto seu funcionamento acontece por meio do que chamamos de componentes (Roveda, 2023). O *React* atualiza com eficiência a interface do usuário, renderizando os componentes relevantes para refletir o estado atual, permitindo que alterações nos dados sejam refletidas na aplicação. Essa abordagem facilita o desenvolvimento de interfaces de usuário dinâmicas e responsivas, tornando o *React* uma escolha popular para a construção de aplicativos Web modernos (Rawat; Mahajan, 2020).

Angular é um *framework* de código aberto desenvolvido pelo Google para a criação de aplicativos dinâmicos e interativos da web. Ele se baseia em *TypeScript*, uma linguagem superset do JavaScript, e utiliza uma abordagem orientada a componentes para a construção de

interfaces. O *Angular* oferece uma estrutura sólida para desenvolvimento, incluindo suporte a gerenciamento de estado, roteamento, validação de formulários e muito mais (Batista, 2024).

O *Node.js* é a ferramenta que vai nos entregar a capacidade de interpretar código JavaScript, de maneira bem similar ao navegador. Quando executamos um comando escrito em JavaScript, o *Node.js* interpreta esse comando e faz a sua conversão para a linguagem de máquina a ser executada pelo computador (Bessa, 2023).

De acordo com a documentação oficial, o Single-Spa (2024) é um *framework* que facilita a integração de diversos MFEs JavaScript em uma única aplicação *frontend*. Usar *single-spa* para arquitetar o *frontend* traz muitos benefícios, como a capacidade de utilizar diferentes *frameworks (React, AngularJS, Angular, Ember,* etc.) na mesma página sem precisar atualizá-la.

Webpack é um *module bundler* (empacotador de módulos) para JavaScript, em outras palavras, ele junta os arquivos JS (e também outros formatos) da sua aplicação (seja arquivos seus ou dependências externas) em um arquivo só (ou mais de um), de forma otimizada. Os arquivos são unificados na ordem certa, sem duplicação e podem ser minificados para reduzir o tamanho (Matoso, 2018).

O *Flask* é um *microframework* web que permite criar APIs de forma simples e eficiente. Com o *Flask*, é possível definir as rotas da API, mapeando os *endpoints* para funções que serão executadas quando uma requisição for feita. Além disso, o *Flask* também oferece recursos para validação de dados, autenticação de usuários e tratamento de erros (Awari, 2023). As principais vantagens de utilizar *Flask* são a simplicidade, velocidade e comunidade. Simplicidade, pois somente as bibliotecas necessárias são carregadas no projeto, velocidade está ligada a simplicidade, como menos código para processar a aplicação se torna mais leve, e a comunidade de desenvolvedores solucionam problemas relacionados ao *framework* todos os dias (Kanashiro et al., 2022).

O *Hyperdrive* estabelece a comunicação necessária para acessar a bibliografia e o acervo das bibliotecas das instituições de ensino superior do país que estão armazenados na *blockchain*. Ele atua como uma ponte entre a interface do usuário e a *blockchain*, garantindo que os métodos de consulta e acesso sejam transmitidos de maneira eficiente. Essa comunicação envolve a identificação das bibliografias e acervos desejados por meio de chaves criptografadas, permitindo a recuperação segura e confiável dessas informações diretamente da *blockchain*.

Figueiredo (2018) destaca que, na prática, o *blockchain* atua como um extenso repositório de dados imutável e compartilhado, projetado para conduzir transações no ambiente digital. Cada interação entre os usuários é registrada em um bloco, que é validado por meio de assinaturas digitais para garantir sua integridade. Cada bloco é identificado de forma única e incorporado em uma sequência de blocos semelhantes, dispostos em uma ordem linear e cronológica. Cada registro produz uma chave criptografada composta de caracteres alfanuméricos, garantindo a segurança das informações armazenadas nos blocos. Os dados também são replicados em numerosos computadores, e essa distribuição descentralizada contribui significativamente para a segurança das transações e dos dados.

O conceito principal por trás do ARK é que o processo de atribuição de identificadores é gerenciado de forma autosuficiente. Não é necessário uma autoridade central para gerar um novo identificador, principalmente porque o ARK não precisa de informações externas para atribuir um novo identificador único. Os *Archival Resource Keys* (ARKs) são um sistema de PID aberto que fornece referências confiáveis para objetos de informação. Os ARKs são amplamente utilizados e já foram adotados por mais de 900 organizações. Essas organizações usam o ARK para gerar mais de 8 bilhões de identificadores, abrangendo desde objetos digitais (documentos e bancos de dados) até objetos físicos (amostras biológicas e obras de arte), seres vivos (pessoas e orquestras) e objetos intangíveis (lugares e termos de vocabulário) (Segundo et al., 2023).

A identificação de objetos em mídias digitais e físicas tornou-se essencial para a comunicação acadêmica em muitos domínios do conhecimento. Um método para identificar entidades em várias fontes é atribuir identificadores únicos de objetos e usá-los posteriormente para se referir a essas entidades em diferentes sistemas. Esses identificadores são chamados Identificadores Persistentes (PID). No contexto científico, os PIDs são essenciais para diversos aspectos, como citação, crédito e autoria, pedidos de patentes, procedência do conhecimento e validação. No ecossistema de pesquisa atual, os PIDs (por exemplo, DOIs) são fundamentais para identificar artigos, livros, autores, documentos, arquivos, bancos de dados, amostras, objetos de arte e páginas, entre outros objetos digitais científicos, em ecossistemas de ciência aberta nacionais, regionais e globais (Segundo et al., 2023).

Segundo et al. (2023) elenca que o dARK é uma extensão do conceito ARK projetada para um ambiente descentralizado. Ele permite que um consórcio de instituições em um ambiente baseado em *blockchain* atribua identificadores persistentes a objetos digitais. O dARK segue os princípios do ARK, permitindo que as instituições gerem identificadores únicos e associem metadados descritivos a esses identificadores. Além disso, o dARK atua como um intermediário para sistemas de PID, como *Digital Object Identifier* (DOI) e *Original Researcher and Contributer ID* (ORCID), e oferece recursos de consulta confiáveis. Essa abordagem descentralizada é projetada para superar limitações dos sistemas de PID convencionais e promover a interoperabilidade em ambientes de pesquisa.

Brito e Valente (2020) esclarece que o REST, é um estilo arquitetural para sistemas distribuídos, estabelece restrições para aprimorar o desempenho e escalabilidade. Ele se baseia no modelo cliente-servidor tradicional e é adotado por APIs REST, que seguem essas restrições. Além disso, REST cria uma interface uniforme com base na identificação de recursos, fornecendo dados dinâmicos por meio de *endpoints*. Cada *endpoint* retorna informações sobre um recurso com um conjunto de campos predefinidos. Li (2011) cita, contudo, alguns especialistas sustentam que a arquitetura REST continuará sendo relevante por um longo período e argumentam que, com a aplicação cuidadosa de técnicas e a adoção de boas práticas, é viável desenvolver APIs robustas e altamente escaláveis com base no modelo REST.

2.3 Trabalhos Relacionados

Diversas abordagens arquiteturais surgiram no cenário tecnológico com o intuito de facilitar a implementação de interfaces que interagem com serviços e microsserviços na internet. Foram analisados oito trabalhos que engloba uma variedade de aspectos, incluindo as distintas arquiteturas empregadas no desenvolvimento de sistemas web, a metodologia adotada por essas estruturas e os benefícios inerentes à aplicação da arquitetura de MFEs. A escolha desses trabalhos foi baseada em sua relevância acadêmica e prática, visando proporcionar uma compreensão abrangente das características e potenciais vantagens proporcionadas pela arquitetura de MFEs no contexto do desenvolvimento de *software*.

O estudo conduzido por Bastos (2020) busca esclarecer a adoção da abordagem de MFEs no desenvolvimento de *software* web, examinando as circunstâncias em que essa abordagem pode oferecer benefícios específicos em comparação com outras alternativas. A pesquisa analisa organizações que incorporaram MFEs em seus processos de desenvolvimento. Nascimento e Sotto (2020) explorou o conceito e a implementação da arquitetura de MFEs, destacando as vantagens e desafios envolvidos ao empregá-la no desenvolvimento de *software*, em contraste com o modelo monolítico convencional em vigor atualmente. Foram desenvolvidas pequenas aplicações usando os *frameworks ReactJS, Angular e Vue.js*.

Filho (2021) aborda a evolução até a arquitetura de MFEs, implementações da arquitetura desmistificando todo o processo de comando de criação do projeto, dependencias e *scripts* do projeto, estrutura de pastas e arquivos, configuração de rotas com *Single SPA* e geração de pacote com *Webpack*. A principal meta por trás da concepção desses protótipos é compreender o processo de criação e configuração de um projeto ao empregar a arquitetura de MFEs, comparativamente à arquitetura monolítica. Queiroz (2023) tem como propósito principal investigar a abordagem integrada na implementação de MFEs por meio do uso de *Web Components e Module Federation*. A ênfase está na criação de uma interface que não esteja vinculada a um *framework frontend* específico, buscando assim flexibilidade e independência na escolha de *frameworks* em ambos os lados do desenvolvimento.

Almeida (2021) analisa a avaliação da arquitetura de MFEs para grandes aplicações web, comparando-a com o modelo tradicional monolítico. A proposta é considerar o website ou aplicativo como uma composição de recursos desenvolvidos por equipes independentes, cada uma especializada em áreas específicas. O estudo destaca benefícios, desvantagens e custos monetários associados à implementação dessa abordagem. Tem como objetivo também buscar identificar as condições ideais para a aplicação de MFEs em contraposição às abordagens tradicionais, permitindo uma avaliação prática da arquitetura proposta.

Wang et al. (2020) aborda o plano *"Education Informatization 2.0"* do Ministério da Educação que impulsionou a criação de sistemas de informação universitários na China, baseados em arquiteturas tradicionais de MVC. Enfrentando desafios decorrentes da complexidade e alto acoplamento entre os negócios educacionais, esses sistemas demandam extenso desenvolvimento de código. Para superar essas questões, propõe-se e implementa-se com sucesso uma solução

de MFEs na plataforma de informações de pós-graduação da Universidade Normal do Leste da China. Alinhada à arquitetura orientada a serviços, essa abordagem destaca-se pela eficácia no desenvolvimento ágil, eficiência na separação de serviços e capacidade de atualizações incrementais. Concluindo-se que a arquitetura de MFEs adapta-se às necessidades futuras dos sistemas de informação de gestão educacional, oferecendo uma perspectiva inovadora para o desenvolvimento de futuras gerações de sistemas educacionais.

Considerando a análise desses desafios e alinhando-se à tecnologia correspondente de *frontend*, o artigo de Yang, Liu e Su (2019) integra o conceito de microsserviços no desenvolvimento *frontend* e apresenta um plano de design para um sistema de gestão de conteúdo baseado em MFEs. Adicionalmente, são minuciosamente delineadas as principais questões práticas, incluindo o conceito de design de MFEs e os métodos de implementação. Rappl (2021) traz o conceito de MFEs incorporando os princípios dos microsserviços, como isolamento de código, separação de responsabilidades e escalabilidade, para o *frontend*. Apresentando alternativas arquiteturais para a construção de MFEs e as compara usando um catálogo de critérios e uma aplicação de exemplo.

Esses motivos deram o incentivo para a elaboração deste trabalho, que se configura como uma alternativa no domínio de arquiteturas de MFEs. O enfoque principal é proporcionar uma abordagem ágil e eficiente, permitindo desenvolvimento independente e reutilizável de diferentes partes da aplicação. A arquitetura escolhida almeja manter-se alinhada com as melhores práticas e padrões modernos, com a finalidade de oferecer uma solução flexível para as demandas presentes e futuras.

3 ASPECTOS METODOLÓGICOS

3.1 Metodologia

Tendo em vista que este trabalho também possui o objetivo de realizar um estudo bibliográfico sobre os conceitos de MFEs e métodos de desenvolvimento de sistemas usando essa arquitetura, realizou-se uma análise sobre os conceitos e soluções existentes de interfaces que consomem serviços e microsserviços na internet, e o desenvolvimento para endereçar o problema do contexto do trabalho. As atividades realizadas durante o trabalho de conclusão do curso foram:

- **Revisão Bibliográfica**: Realização de estudos de bibliografias sobre os conceitos e soluções existentes para uma arquitetura de *software* e *frontend* baseada em MFEs;
- **Elicitação de Requisitos**: Identificação e construção dos requisitos referentes aos protótipos dentro do projeto de interface;
- **Desenvolvimento do *Frontend***: Implementação de uma aplicação *frontend* com interface responsiva e intuitiva para consumir e prover dados à API;
- **Validação e Verificação**: Realização de testes para verificar se o *frontend* está acessando corretamente os *endpoints* necessários na API.

3.1.1 Tipologia da pesquisa

Todo bom trabalho de desenvolvimento surge de uma pesquisa de literatura aprofundada. Wazlawick (2021) recomenda que todo trabalho de pesquisa comece com uma revisão bibliográfica, mapeando a literatura para entender o panorama geral e evitar esforços redundantes. Uma revisão bibliográfica sistemática é essencial para responder a questões profundas sobre os resultados das pesquisas, garantindo que as contribuições sejam novas e relevantes, evitando duplicidade e facilitando a comparação com trabalhos existentes.

Esta pesquisa pode ser classificada quanto à natureza, objetivo e procedimentos técnicos de acordo com as ideias de Wazlawick (2021). Quanto à natureza, a pesquisa se classifica como primária, pois seu objetivo principal é desenvolver novos conhecimentos e soluções para abordar a falta de uma camada de *frontend* que integre com uma API criada para simular o funcionamento da API baseada em *blockchain*. Em vez de se concentrar apenas na revisão e análise de informações existentes, o estudo envolve a realização de desenvolvimento de protótipos e implementar a arquitetura mais adequada para a aplicação.

Ela também se classifica como pesquisa descritiva devido ao foco em descrever a situação da ausência de uma camada de *frontend* para integrar com uma API fictícia. A abordagem envolveu categorizar as informações coletadas e oferecer uma análise objetiva dos fatos existentes, sem interferir diretamente ou buscar formular teorias explicativas. As etapas de levantamento

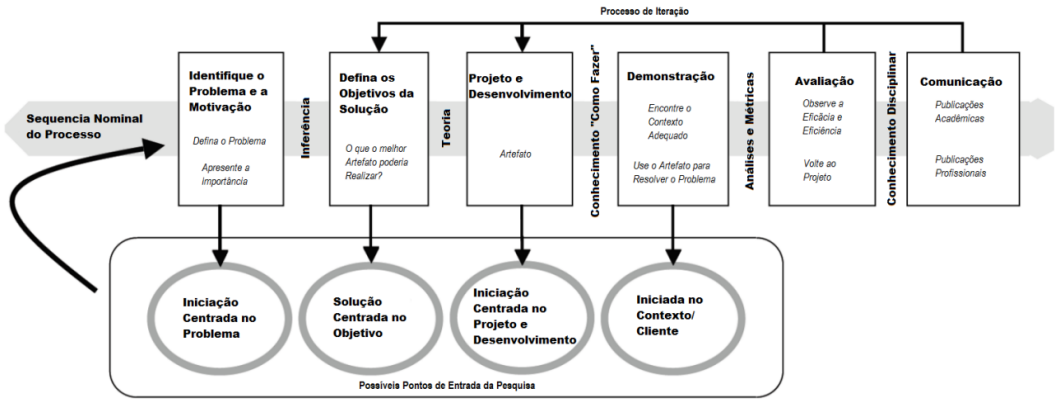
de dados, revisões de literatura e análise de requisitos foram cruciais, destacando a natureza descritiva da pesquisa.

O trabalho também possui características de pesquisa exploratória, pois analisa diferentes abordagens arquiteturais e os desafios enfrentados por outros autores ao utilizá-las. Ao mostrar trabalhos relacionados, descreve-se os desafios que outros autores enfrentaram ao usar a arquitetura de MFEs, a qual foi escolhida para o desenvolvimento do sistema.

No contexto dos procedimentos técnicos, o trabalho caracteriza-se como uma pesquisa bibliográfica aplicada ao desenvolvimento de projeto. Utiliza-se a pesquisa bibliográfica para analisar conceitos, teorias e soluções existentes no campo de desenvolvimento de interfaces. A partir dessa análise, seleciona-se e justifica-se a melhor arquitetura de interface para uma aplicação específica. Essa abordagem combina a revisão teórica detalhada com a aplicação prática, garantindo que as decisões sejam embasadas em evidências sólidas e voltadas para resolver problemas concretos de design e usabilidade. Assim, o trabalho não apenas compreende as práticas existentes, mas também propõe a aplicação prática desses conhecimentos para desenvolver uma solução eficaz e inovadora. É importante ressaltar que essa parte bibliográfica teve foco exploratório para o desenvolvimento das interfaces.

De acordo com Dresch, Lacerda e Junior (2015) a pesquisa em *Design Science Research* (DSR), também conhecida como pesquisa construtiva, é uma metodologia que visa desenvolver artefatos capazes de oferecer benefícios práticos às pessoas. Este método, ilustrado na Figura 2, envolve a criação de inovações com o propósito específico de solucionar problemas do mundo real. Além de resolver esses desafios, o objetivo é contribuir para o avanço científico ao oferecer soluções prescritivas que possam ser aplicadas de maneira eficaz.

Figura 2 – *Design Science Research*



Fonte: Traduzido de Peffers et al. (2007).

Dresch, Lacerda e Junior (2015) destacam várias etapas comuns entre os diferentes métodos de pesquisa. Essas etapas incluem a definição do problema, a revisão da literatura e a busca por teorias existentes, a proposição de possíveis soluções, o desenvolvimento, a avaliação, a decisão sobre a melhor solução, a reflexão e o aprendizado, e a comunicação dos resultados.

3.2 Desafios da arquitetura de MFEs

Peltonen, Mezzalira e Taibi (2021) cita que os MFEs nem sempre se adequam a todos os tipos de aplicação devido à sua natureza e à possível complexidade que podem introduzir nos aspectos técnicos e organizacionais do projeto. A complexidade aumenta consideravelmente, especialmente quando se lida com um grande número de clientes com necessidades significativamente diversas. A integração de vários subprojetos em uma única aplicação pode se tornar complicada, transformando-se em algo semelhante a uma arquitetura distribuída. Os MFEs naturalmente adicionam mais pontos de gestão ao projeto. A implementação desse tipo de arquitetura requer uma análise inicial mais aprofundada para compreender como a integração ocorrerá e como a aplicação pode ser dividida em pequenos módulos.

Conforme os sistemas vão escalando e consequentemente a base de código vai acompanhando esse crescimento. Manter uma arquitetura escalável torna-se cada vez mais necessária, porém é um processo complicado e desafiador. As aplicações grandes trazem naturalmente uma complexidade maior, o que muitas vezes dificultam o entendimento do funcionamento de toda a aplicação, seja para os novos desenvolvedores de *software* e até mesmo para os mais experientes, impactando diretamente na produtividade, no trabalho em equipe, qualidade de entrega, entre outros (Nascimento; Sotto, 2020).

Filho (2021) observou durante o avanço do projeto, um aumento na complexidade das regras de negócio e na expansão da base de código. Tornando a incorporação de novas funcionalidades ou a manutenção do código existente uma tarefa desafiadora, devido à interdependência entre as diversas partes do código. Essa ampliação também implica em uma demanda crescente de recursos computacionais, resultando em uma desaceleração do trabalho dos engenheiros. Durante o desenvolvimento dos protótipos, foi notável o incremento de complexidade proporcionado pela arquitetura de MFEs. A configuração e a construção do projeto demandam consideravelmente mais esforço em comparação à arquitetura monolítica, exigindo uma utilização mais extensa de bibliotecas e *frameworks*. Um dos objetivos gerais no desenvolvimento de *software* é minimizar a redundância de código. No entanto, como os MFEs englobam várias equipes independentes desenvolvendo seus projetos paralelamente, isso pode introduzir muita redundância de código JavaScript e CSS. Isso desnecessariamente acaba aumentando a quantidade de código da aplicação (Prajwal; Parekh; Shettar, 2021).

4 RESULTADOS E DISCUSSÕES

Neste trabalho o sistema web desenvolvido tem como objetivo principal criar interfaces, adotando uma arquitetura de MFEs. Para embasar esse processo, foi conduzida uma pesquisa bibliográfica abrangente sobre a arquitetura MFEs, explorando seus conceitos e soluções, fornecendo a base teórica necessária para o desenvolvimento. Este *frontend* foi desenvolvido para validar os conceitos da arquitetura de micro *frontends*, operando por via de requisições em uma API fictícia que simula um serviço baseado em *blockchain*.

A escolha dessa arquitetura se mostra ideal para lidar com esse cenário, pois proporciona uma abordagem inovadora e flexível. Durante o desenvolvimento das interfaces, foi enfrentado um desafio considerável de integração com a API REST *Hyper Drive* baseada em *blockchain* devido à ausência de respostas de saída do backend. Essa limitação inicialmente dificultou a realização de testes completos e a validação da funcionalidade esperada da aplicação utilizando um ambiente real.

Para contornar essa situação, foi decidido simular as interações com a API utilizando uma API fictícia que simula o comportamento esperado da API baseada em *blockchain*. Essa abordagem permitiu continuar o desenvolvimento da camada *frontend* e realizar testes das interações planejadas, mantendo o formato e a estrutura das requisições que seriam feitas na API real.

A validação e verificação foram realizadas por meio da execução de testes destinados a garantir que o *frontend* está acessando corretamente os *endpoints* da API fictícia. Esses testes foram realizados utilizando o *Mockable*, uma ferramenta que permite simular requisições como se fossem feitas para a API real. Por meio dessa abordagem, foi possível avaliar a funcionalidade do *frontend* em um ambiente controlado, assegurando que ele interage adequadamente com os serviços esperados da API

É importante destacar que, mesmo diante das limitações enfrentadas, o projeto foi inicialmente planejado para integrar-se com a API real baseada em *blockchain*. A integração com a API real pode ser implementada a qualquer momento, uma vez que o *backend* esteja operacional. E os resultados obtidos por meio da simulação da API fictícia foram satisfatórios para demonstrar o funcionamento da aplicação dentro do ambiente simulado.

A escolha da estrutura de MFEs proporcionou uma implementação mais ágil e simplificada do desenvolvimento web, graças aos padrões de comunicação claros e interfaces bem estruturadas entre os componentes. Isso resultou em redução de interferências e gargalos, oferecendo modularidade e escalabilidade, facilitando a colaboração e permitindo uma expansão incremental suave. A flexibilidade oferecida pela arquitetura também permitiu a escolha das tecnologias mais adequadas para cada parte da aplicação, garantindo uma adaptação eficiente às mudanças e ao crescimento futuro.

Durante o desenvolvimento deste projeto, foi utilizado um *software* licenciado sob uma licença permissiva, especificamente a licença MIT. A utilização dessa licença permitiu

o uso, modificação e redistribuição do *software* original, o que se mostrou essencial para a implementação das funcionalidades necessárias para este trabalho. No contexto deste projeto, o mesmo foi utilizado como base para o desenvolvimento de novas funcionalidades, adaptadas para atender às necessidades particulares do sistema web proposto. Todas foram realizadas em conformidade com os termos da licença MIT. O aviso de direitos autorais original e o aviso de permissão foram mantidos, garantindo o cumprimento das exigências legais.

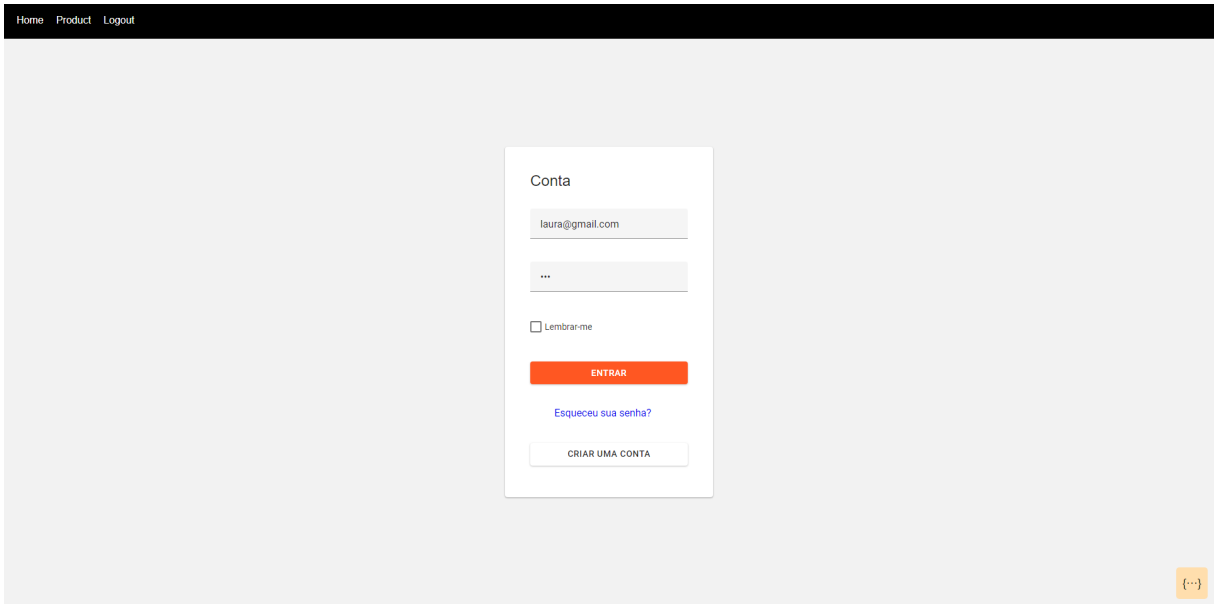
Na prática, as telas foram implementadas com diferentes tecnologias de acordo com as necessidades específicas de cada funcionalidade. O Sistema de *login* foi desenvolvido com *Angular* para garantir a autenticação dos usuários, enquanto a *Home* foi criada em *React* e é acessível sem autenticação. A *Navbar*, também em *React*, é responsável por exibir a barra de navegação, e a página de Produto em *React* que gerencia as requisições de dados.

O usuário pode buscar por título, nome ou PID, redirecionando para páginas com detalhes como link do artigo, revista ou livro. O usuário também pode acessar a página para fazer requisições clicando no botão na tela de Produto. Tanto o *Root-config* quanto o *Root-config.html* atuaram como *shells* para hospedar os micro *frontends*, garantindo uma integração fluida entre os componentes da aplicação. A integração entre os princípios teóricos da arquitetura de MFEs e sua aplicação prática na construção deste sistema web evidencia os benefícios dessa abordagem para o desenvolvimento de interfaces dinâmicas e escaláveis.

4.1 Organização do Projeto

Na Figura 3 é apresentado o sistema de *login* do projeto desenvolvido utilizando *Angular*. Dentro da estrutura principal, estão os componentes responsáveis pelo formulário de autenticação e pelo *logout*, juntamente com o serviço que gerencia a sessão do usuário. O projeto é integrado ao *single-spa* para suportar uma arquitetura de MFEs.

Figura 3 – *Login*



Fonte: Elaborada pelo autor.

As dependências listadas na Figura 4 incluem várias bibliotecas do *Angular*, essenciais para a construção de aplicações web dinâmicas. Elas abrangem módulos para animações, formulários, roteamento, entre outros. Há também pacotes específicos como *devextreme e devextreme-angular* para componentes de interface do usuário, *rxjs* para programação reativa, e *single-spa com single-spa-angular* para suportar a arquitetura de MFEs. *Tslib e zone.js* são incluídos para suporte ao *TypeScript* e gerenciamento de zonas no *Angular*, respectivamente.

Figura 4 – Dependências *Login*

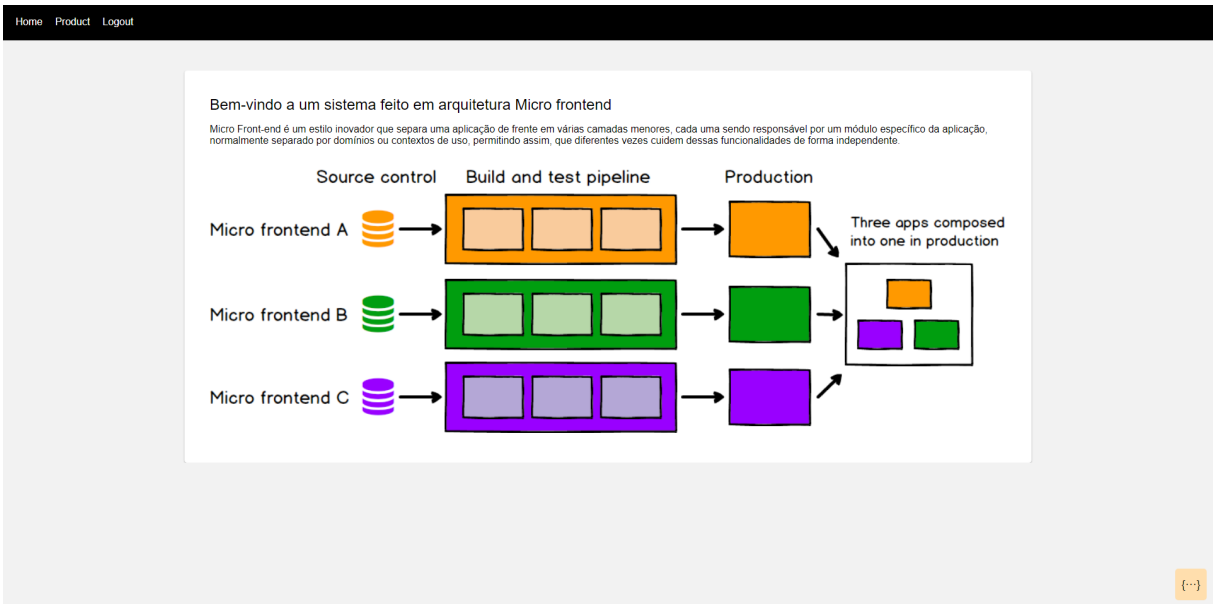
```
"dependencies": {
  "@angular/animations": "^14.2.0",
  "@angular/common": "^14.2.0",
  "@angular/compiler": "^14.2.0",
  "@angular/core": "^14.2.0",
  "@angular/forms": "^14.2.0",
  "@angular/platform-browser": "^14.2.0",
  "@angular/platform-browser-dynamic": "^14.2.0",
  "@angular/router": "^14.2.0",
  "@ngneat/until-destroy": "^9.2.2",
  "devextreme": "^22.2.3",
  "devextreme-angular": "^22.2.3",
  "rxjs": "~7.5.0",
  "single-spa": ">=4.0.0",
  "single-spa-angular": "^7.1.0",
  "tslib": "^2.3.0",
  "zone.js": "~0.11.4"
},
```

Fonte: Elaborada pelo autor.

A pagina inicial *Home* exibida na Figura 5, consiste apenas em uma foto representativa do conceito de MFEs. Não há funcionalidades nesta página, serve apenas como uma introdução

visual ao tema abordado. No diretório *Home*, os arquivos são organizados de forma clara entre os de distribuição e os de origem. Nos arquivos de distribuição, encontramos os resultados finais da construção da aplicação, enquanto nos de origem estão os componentes, estilos e lógica, junto com os arquivos de configuração para facilitar o desenvolvimento. O arquivo de registro de dependências exatas do projeto é mantido, assim como o arquivo que lista as dependências, *scripts* e outras informações importante.

Figura 5 – *Home* (Uso de *React*)



Fonte: Elaborada pelo autor.

As dependências do *Home* na Figura 6, incluem ferramentas fundamentais para desenvolvimento, como *Webpack* e *TypeScript*, bibliotecas *React* para construção de interfaces e *React Router* para roteamento. Além disso, há dependências específicas do contexto de MFEs, como *single-spa* para integração de MFEs e *devextreme* para componentes de interface.

Figura 6 – Dependências *Home*

```
},
"dependencies": {
  "@types/jest": "^27.0.1",
  "@types/systemjs": "^6.1.1",
  "@types/webpack-env": "^1.16.2",
  "devextreme": "^22.2.3",
  "devextreme-react": "^22.2.3",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-router-dom": "^6.6.1",
  "sass": "^1.57.1",
  "single-spa": "^5.9.3",
  "single-spa-react": "^4.3.1",
  "url-loader": "^4.1.1"
},
```

Fonte: Elaborada pelo autor.

O *Product* é uma aplicação *frontend* com uma estrutura organizada em diversos diretórios e arquivos. No diretório principal, componentes como *product-details, product-list e product-pid* lidam com a exibição de detalhes de produtos específicos e listagem dos produtos disponíveis dos projetos científicos. Fora do diretório principal, existem arquivos de configuração importantes que contribuem para o funcionamento da aplicação.

Para o diretório *Product* foram usadas várias dependências essenciais. Na Figura 7 é apresentado o *devextreme* e *devextreme-react* para componentes UI avançados. As bibliotecas *react* e *react-dom* foram primordiais para a construção e renderização dos componentes, os dois gerenciando a navegação entre páginas. *Single-spa e single-spa-react* foram utilizados para a construção de MFEs, permitindo que várias aplicações *frontend* funcionem de forma independente. Por fim, *url-loader* foi utilizado para importar arquivos como URLs, o que é útil para gerenciar *assets* como imagens.

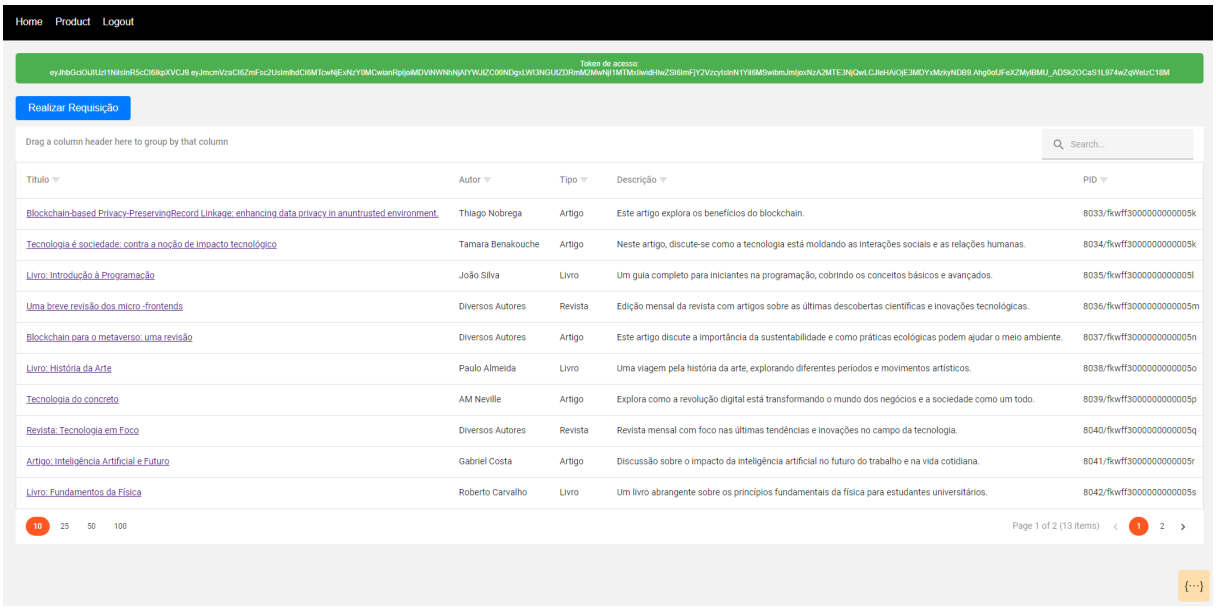
Figura 7 – Dependências *Product*

```
},
"dependencies": {
  "devextreme": "^22.2.3",
  "devextreme-react": "^22.2.3",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-router-dom": "^6.6.1",
  "single-spa": "^5.9.3",
  "single-spa-react": "^4.3.1",
  "url-loader": "^4.1.1"
},
```

Fonte: Elaborada pelo autor.

O *product-list* exibe uma tabela contendo informações essenciais conforme ilustrado na Figura 8, como título, autor, descrição, tipo e PID, tornando mais fácil para os usuários encontrarem livros, artigos ou revistas. Acima da tabela, é exibido um token de acesso, fornecido ao usuário após o *login* no sistema, garantindo a autenticação. Adicionalmente, há um botão "Realizar Requisição", permitindo aos usuários executarem ações específicas relacionadas as requisições.

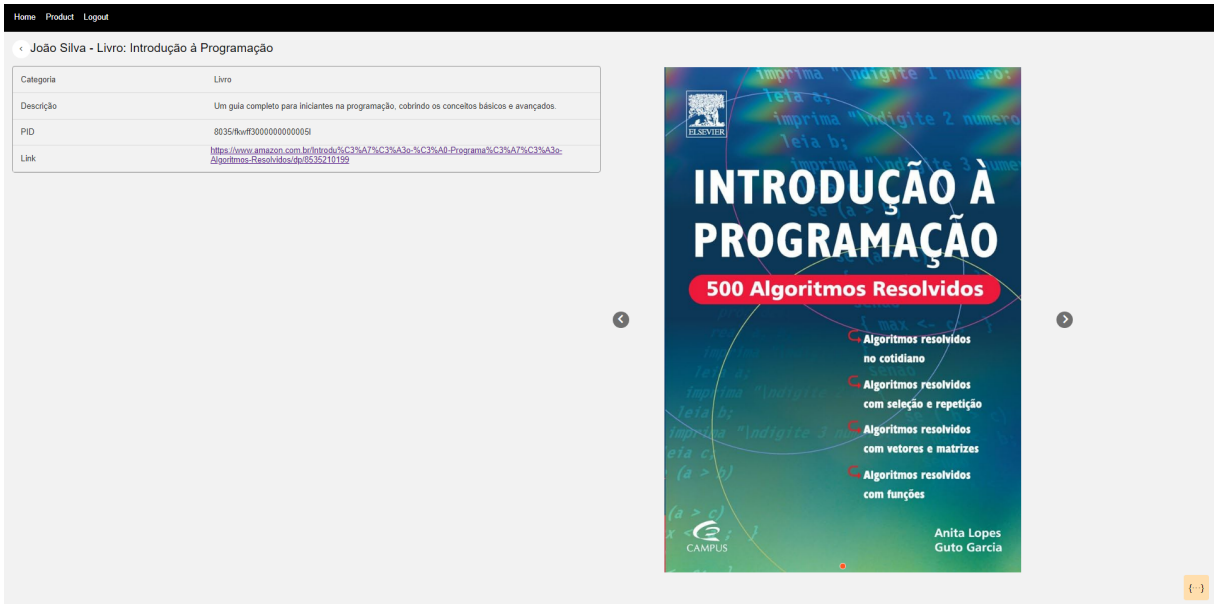
Figura 8 – *Product-List* (Uso de *React*)



Fonte: Elaborada pelo autor.

Quando um usuário clica em um título de interesse na página de *product-list*, ele é redirecionado para a página *product-details*, onde encontra informações mais detalhadas sobre o produto selecionado. A partir das informações já apresentadas anteriormente, como título, autor, descrição e PID, a página *ProductDetails* como mostrado na Figura 9, também inclui detalhes adicionais, como o tipo de pesquisa ao qual o produto está relacionado, um link para o tipo de pesquisa associado ao produto e, se o produto for um livro, uma foto representativa do livro.

Figura 9 – *Product-Details* (Uso de *React*)

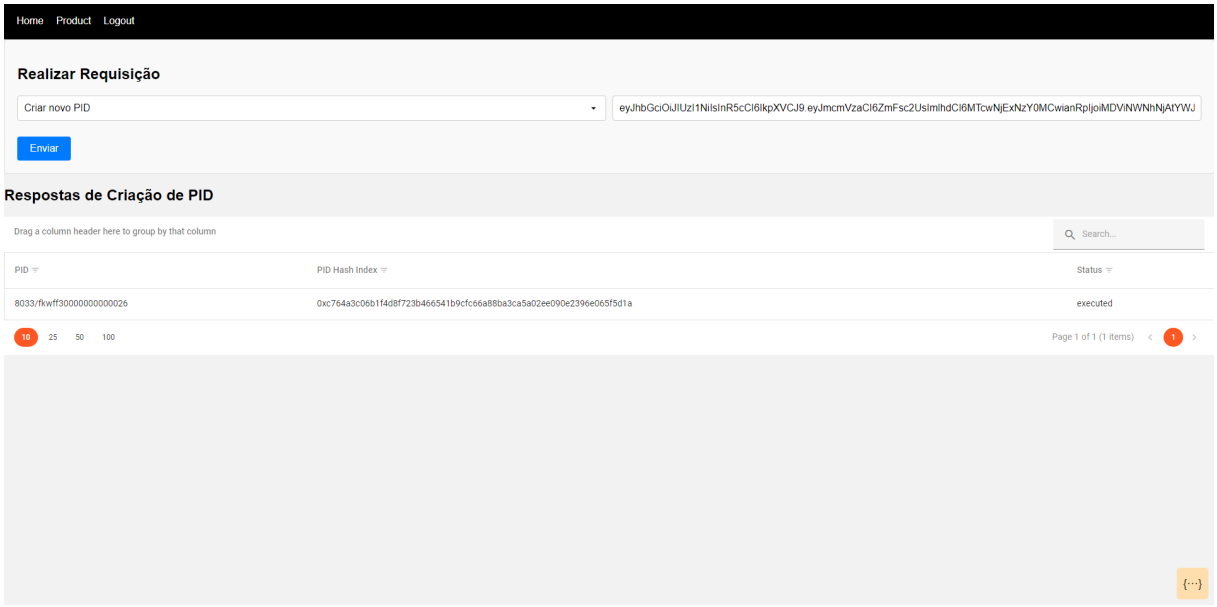


Fonte: Elaborada pelo autor.

Ao clicar no botão “Realizar Requisição” na página *product-list*, o usuário é redirecionado a uma página para efetuar a requisição. Nesta página, ele tem a opção de criar um novo PID (Figura 10), utilizando o token de acesso para autenticação. Além disso, ele pode adicionar uma URL ou PID (Figura 11) e (Figura 12), passando um parâmetro por vez, permitindo adicionar uma URL ao PID sendo criado.

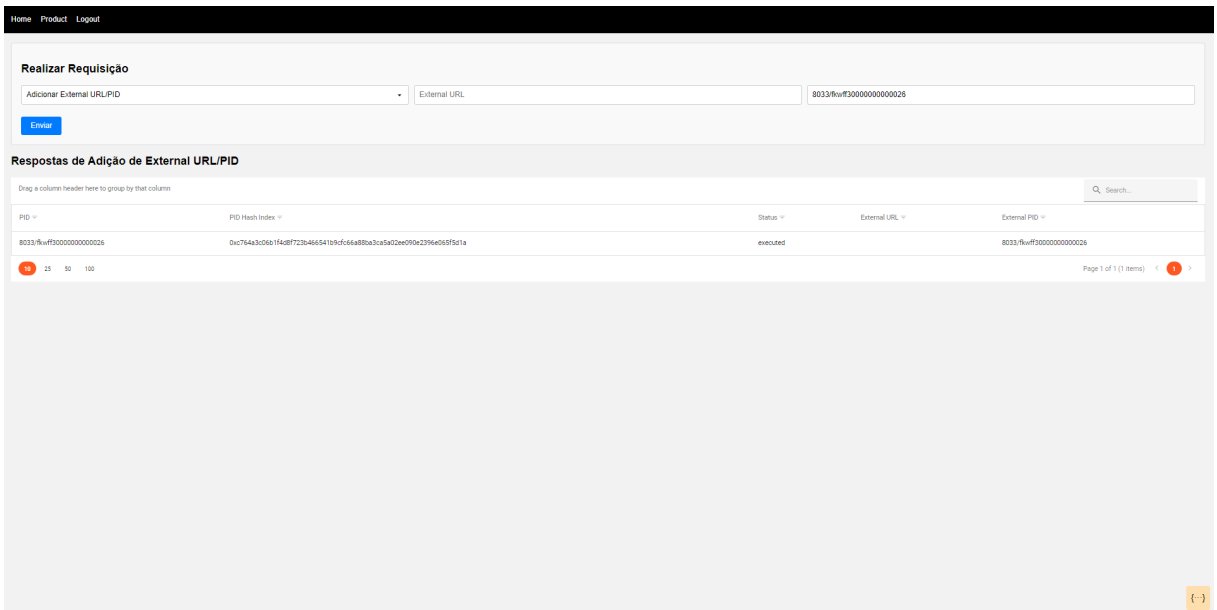
O usuário também pode atribuir atributos ao PID (Figura 13) , utilizando o DOI, autor, título e tipo. Esses atributos adicionados serão então exibidos na tabela de *product-pid*, possibilitando uma visão ampla e organizada dos produtos disponíveis na biblioteca. Essa funcionalidade permite aos usuários gerenciar informações dos produtos de forma dinâmica e conveniente.

Figura 10 – Criar novo PID (Uso de *React*)



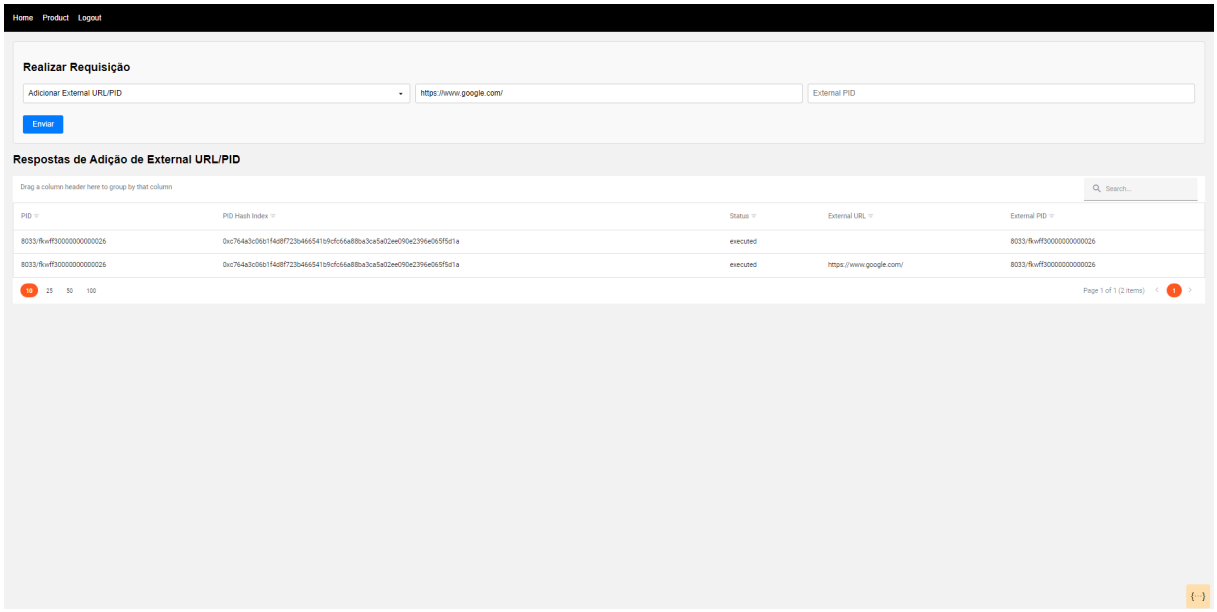
Fonte: Elaborada pelo autor.

Figura 11 – Adicionar *externalPid* ao PID (Uso de *React*)



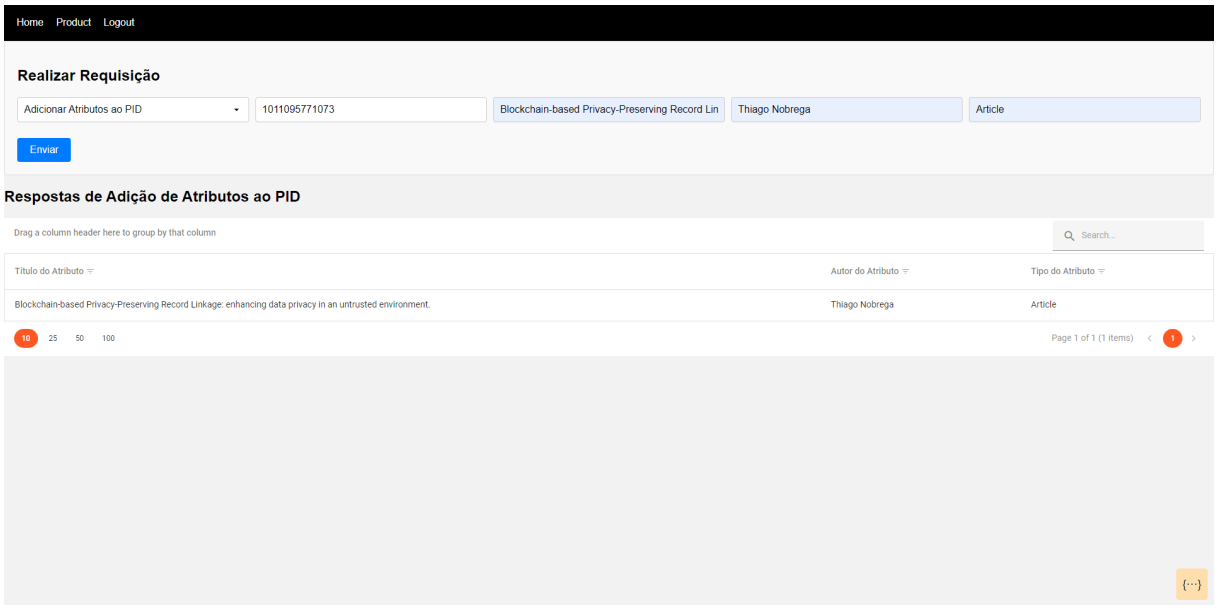
Fonte: Elaborada pelo autor.

Figura 12 – Adicionar URL ao PID (Uso de *React*)



Fonte: Elaborada pelo autor.

Figura 13 – Adicionar Atributos ao PID (Uso de *React*)



Fonte: Elaborada pelo autor.

No diretório *Navbar* (Figura 14) o componente em *React* define a estrutura principal da interface de usuário. Ele inclui um arquivo CSS para estilização e utiliza o *React Router* para gerenciar a navegação entre diferentes páginas. Dentro do componente, há uma barra de navegação com três links: *Home, Product e Logout*. Cada link direciona o usuário para uma página específica da aplicação.

Figura 14 – *Navbar* (Uso de *React*)

Fonte: Elaborada pelo autor.

No desenvolvimento do projeto, foram utilizadas as dependências necessárias. Na Figura 15 mostra a estrutura que contem definições de tipo para o Jest, garantindo a tipagem correta ao escrever testes em *TypeScript*. O *@types/react e @types/react-dom* ajudam a manter a segurança de tipo ao desenvolver componentes *React* e manipular o DOM. *@Types/systemjs e @types/webpack-env* garantem a tipagem correta ao usar *SystemJS* e variáveis globais do *Webpack*, respectivamente.

A biblioteca *React* foi usada para construir a interface de usuário, com *react-dom* para renderizar componentes no DOM e *react-router-dom* para gerenciar a navegação. *Single-spa* foi empregado para construir MFEs, permitindo integrar várias aplicações de forma independente, enquanto *single-spa-react* facilitou a construção dos MFEs com *React*.

Figura 15 – Dependências *Navbar*

```
},
"dependencies": {
  "@types/jest": "^27.0.1",
  "@types/react": "^17.0.19",
  "@types/react-dom": "^17.0.9",
  "@types/systemjs": "^6.1.1",
  "@types/webpack-env": "^1.16.2",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-router-dom": "^6.6.1",
  "single-spa": "^5.9.3",
  "single-spa-react": "^4.3.1"
},
```

Fonte: Elaborada pelo autor.

O *root-config* foi implementado utilizando um modelo EJS e serve como *shell* para hospedar todos os aplicativos MFEs. O *root-config-html*, em JS e HTML, também funciona como *shell* para hospedar os aplicativos *Home, Produto e Login*. Na Figura 16 são apresentadas diversas dependências do *root-config*, incluindo ferramentas para testes, configuração de tipos, roteamento, e integração com outras bibliotecas. Essas dependências são essenciais para garantir o funcionamento adequado e eficiente do projeto, oferecendo recursos como manipulação de tipos, execução de testes automatizados e roteamento de páginas.

Figura 16 – Dependências *Root-config*

Fonte: Elaborada pelo autor.

No diretório *root-config*, há arquivos fundamentais, como declarações de tipos em *TypeScript*, um arquivo para gerar HTML dinamicamente, um para definir o *layout* básico da aplicação de MFEs, e outro para configurar o *shell* principal. Estão localizados também os arquivos de dependências do projeto, um arquivo que serve como ponto de entrada principal, e arquivos que registram a árvore de dependências e o manifesto do projeto.

Para executar a aplicação de MFEs, é necessário instalar algumas dependências mostradas na Figura 17, essas dependências estão no *package.json* do projeto geral:

- ***Install***: O primeiro install é responsável por instalar todas as dependências do projeto em série;
- ***Install:* <diretório específico>**: Instala as dependências de cada MFE *(root-config, auth, navbar, home, product)*;
- ***Start***: Inicia todos os MFEs simultaneamente;
- ***Start:* <diretório específico>**: Inicia cada MFE individualmente *(root-config, auth, navbar, home, product)*.

Figura 17 – *Package.json*

```
"scripts": {
  "install": "npm-run-all --serial install:*",
  "install:root-config": "cd root-config && npm install",
  "install:root-config-html": "cd root-config-html && npm install",
  "install:auth": "cd auth && npm install",
  "install:navbar": "cd nav-bar && npm install",
  "install:home": "cd home && npm install",
  "install:product": "cd product && npm install",
  "start": "npm-run-all --parallel start:*",
  "start:root-config": "cd root-config && npm start",
  "start:root-config-html": "cd root-config-html && npm start",
  "start:auth": "cd auth && npm start",
  "start:navbar": "cd nav-bar && npm start",
  "start:home": "cd home && npm start",
  "start:product": "cd product && npm start"
},
```

Fonte: Elaborada pelo autor.

O arquivo de *Import Map Overrides* ilustrado na Figura 18, desempenha um papel indispensável ao definir como os módulos JavaScript são importados e tratados durante a execução do aplicativo. Essas configurações permitem personalizar o carregamento de módulos para diferentes domínios e MFEs, garantindo uma experiência de usuário eficiente e coesa. Por meio dessas substituições e sobrescritas de importações, é possível adaptar o comportamento de roteamento e carregamento de módulos para corresponder às necessidades específicas do projeto. Isso demonstra a flexibilidade e poder da arquitetura de MFEs em lidar com complexidades de integração e personalização, resultando em um ecossistema de desenvolvimento altamente adaptável.

Figura 18 – *Import Map Overrides*

Fonte: Elaborada pelo autor.

Ao longo do desenvolvimento deste trabalho, a autora pôde perceber a aplicação prática dos conhecimentos adquiridos, em que todas as disciplinas cursadas foram essenciais para cada detalhe, especialmente em Web, tendo a oportunidade de desenvolver projetos práticos no *frontend*. A experiência na disciplina não apenas proporcionou uma compreensão dos conceitos, mas também foi fundamental para abrir leques de aprendizados na prática para alcançar os objetivos estabelecidos neste Trabalho de Conclusão de Curso. A partir desses conhecimentos,

a autora se sentiu estimulada a aprofundar seu entendimento sobre tecnologias como *React* e explorar outras tecnologias relacionadas, ampliando assim uma gama de domínios técnicos. Foi na resolução dos problemas encontrados durante o trabalho que a autora obteve os aprendizados mais significativos. Cada obstáculo enfrentado foi uma oportunidade para seu desenvolvimento pessoal e profissional, permitindo aprimorar habilidades de resolução de problemas e a pensar de forma mais estratégica. Nesse sentido, a autora (discente em conclusão de curso) avalia que este trabalho contribuiu imensamente para o seu aprendizado, proporcionando não apenas conhecimento técnico nas tecnologias usadas, mas também experiência prática na implementação de soluções vistas neste trabalho.

5 CONCLUSÃO E TRABALHOS FUTUROS

A implementação de MFEs trouxe uma série de vantagens significativas para o projeto, como maior escalabilidade, facilidade de manutenção e flexibilidade tecnológica. Esta abordagem permitiu que diferentes partes da aplicação fossem desenvolvidas e implantadas de forma independente, facilitando o trabalho de diversas equipes simultaneamente. Além disso, o sistema tornou-se mais preparado para crescer conforme as suas necessidades, podendo integrar novas funcionalidades sem comprometer a estabilidade ou a performance geral. A utilização de tecnologias como *Single-SPA, Node.js, Angular, React e Webpack* evidenciou os benefícios práticos dessa arquitetura, demonstrando que a abordagem de MFEs é uma solução viável e vantajosa para o desenvolvimento de aplicações web dinâmicas e escaláveis.

5.1 Sugestões para Trabalhos Futuros

- Testes de Usabilidade: Realizar testes de usabilidade com usuários finais para avaliar a intuitividade e a eficiência da interface;
- Integrar com uma API Real: Realizar integração com uma API real para validar completamente o comportamento do *frontend*;
- Otimizações e Melhorias Contínuas: Continuar a otimizar o desempenho e a responsividade da aplicação, explorando novas tecnologias e *frameworks*;
- *Feedback* dos Usuários: Coletar *feedback* dos usuários após a implementação para identificar problemas e oportunidades de melhoria;
- Disponibilizar uma seção com tutoriais ou guias rápidos para ajudar os usuários a se familiarizarem com as funcionalidades da aplicação, incentivando o uso e a exploração na página Home;
- Implementar “esquecer a senha” e “criar cadastro” em *Login*.

REFERÊNCIAS

ALMEIDA, D. F. F. *Micro Frontends para Aplicações Web*. 2021. 2023. Citado na página 19.

AWARI. *Api de Python: Aprenda a Utilizar a Linguagem de Programação Mais Popular para Desenvolvimento Web*. 2023. Citado na página 17.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice: Software Architect Practice_c3*. [S.l.]: Addison-Wesley, 2012. Citado na página 12.

BASTOS, J. P. d. S. *Desenvolvimento Web Orientado a Micro Frontends*. Tese (Doutorado), 2020. Citado na página 19.

BATISTA, N. Angular: o que é, para que serve e um guia para iniciar no framework javascript. 2024. Atualizado em 16 de Abril. Disponível em: https://www.alura.com.br/artigos/angular-js. Citado na página 17.

BELLINI, E. A blockchain based trusted persistent identifier system for big data in science. *Foundations of Computing and Decision Sciences*, v. 44, n. 4, p. 351–377, 2019. Citado na página 15.

BESSA, A. Node.js: o que é, como funciona esse ambiente de execução javascript e um guia para iniciar. *Alura*, 2023. Atualizado em 18/09/2023. Disponível em: https://www.alura.com.br/artigos/node-js#:~:text=Runtime%20e%20V8-,O%20Node.,a%20ser%20executada%20pelo%20computador. Citado na página 17.

BRITO, G.; VALENTE, M. T. Rest vs graphql: A controlled experiment. In: IEEE. *2020 IEEE international conference on software architecture (ICSA)*. [S.l.], 2020. p. 81–91. Citado na página 18.

DRESCH, A.; LACERDA, D. P.; JUNIOR, J. A. V. A. *Design science research: método de pesquisa para avanço da ciência e tecnologia*. [S.l.]: Bookman Editora, 2015. Citado 2 vezes nas páginas 12 e 22.

FIGUEIREDO, A. Aumentando a competitividade. *https://fenacon.org.br/wp-content/uploads/2020/12/FENACON*$_1$87$_4$*fEwYFh.pdf*, 2018. *Citadonapgina*17.

FILHO, M. A. A. Arquitetura de micro frontends no desenvolvimento web. 2021. Citado 3 vezes nas páginas 13, 19 e 23.

FLANAGAN, D. *JavaScript: o guia definitivo*. [S.l.]: Bookman Editora, 2012. Citado na página 16.

FLATSCHART, F. *HTML 5-Embarque Imediato*. [S.l.]: Brasport, 2011. Citado na página 16.

GEERS, M. Micro frontends extending the microservice idea to frontend development. *https://micro-frontends.org/*, 2017. Acesso em 10 de outubro de 2023. Citado na página 15.

GOLODONIUC, P.; CAR, N. J.; KLUMP, J. Distributed persistent identifiers system design. *Data Science Journal*, v. 16, p. 34–34, 2017. Citado na página 14.

HAKALA, J. Persistent identifiers-an overview. In: . [S.l.: s.n.], 2010. Citado na página 14.

JACKSON, C. Micro frontends extending the microservice idea to frontend development. *https://martinfowler.com/articles/micro-frontends.html*, 2019. Acesso em 10 de outubro de 2023. Citado 3 vezes nas páginas 11, 13 e 15.

KANASHIRO, A. L. H. et al. Game4code: um sistema auxiliar gamificado para o aprendizado da linguagem de programação java. Cruzeiro do Sul Educacional, 2022. Citado na página 17.

LI, H. Restful web service frameworks in java. In: IEEE. *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*. [S.l.], 2011. p. 1–4. Citado na página 18.

LINDLEY, C. Frontend developer handbook 2017. *Frontend masters*, 2017. Citado na página 11.

MATOSO, D. *Webpack sem Medo — Parte 1: Introdução*. 2018. Disponível em: https://www.webdevdrops.com/webpack-sem-medo-introducao-af889eb659e7/. Citado na página 17.

MCMURRY, J. A. et al. Identifiers for the 21st century: How to design, provision, and reuse persistent identifiers to maximize utility and impact of life science data. *PLoS biology*, Public Library of Science San Francisco, CA USA, v. 15, n. 6, p. e2001414, 2017. Citado na página 14.

NASCIMENTO, C. P.; SOTTO, E. C. S. Microfrontend: um estudo sobre o conceito e aplicação no frontend. *Revista Interface Tecnológica*, v. 17, n. 1, p. 153–165, 2020. Citado 2 vezes nas páginas 19 e 23.

PELTONEN, S.; MEZZALIRA, L.; TAIBI, D. Motivations, benefits, and issues for adopting micro-frontends: a multivocal literature review. *Information and Software Technology*, Elsevier, v. 136, p. 106571, 2021. Citado na página 23.

PRAJWAL, Y.; PAREKH, J. V.; SHETTAR, R. A brief review of micro-frontends. *United International Journal for Research and Technology*, v. 2, n. 8, 2021. Citado na página 23.

QUEIROZ, R. C. P. d. Microfrontends com web components e webpack: Uma abordagem de implementação agnóstica em relação ao framework. 2023. Citado na página 19.

RAPPL, F. *The The Art of Micro Frontends: Build websites using compositional UIs that grow naturally as your application scales*. [S.l.]: Packt Publishing Ltd, 2021. Citado na página 20.

RAWAT, P.; MAHAJAN, A. N. Reactjs: A modern web development framework. *International Journal of Innovative Science and Research Technology*, v. 5, n. 11, p. 698–702, 2020. Citado na página 16.

ROVEDA, U. Front end: O que é, como funciona e qual a importância. *https://www.totvs.com/blog/developers/front-end/*, 2023. Acesso em 10 de outubro de 2023. Este é um site que contém informações sobre conceitos de front end e suas impotâncias. Citado na página 16.

SEGUNDO, W. et al. *dARK: A decentralized blockchain implementation of ARK Persistent Identifiers*. [S.l.], 2023. Citado 2 vezes nas páginas 14 e 18.

SICILIA, M.-A. et al. Decentralized persistent identifiers: a basic model for immutable handlers. *Procedia computer science*, Elsevier, v. 146, p. 123–130, 2019. Citado na página 14.

SINGLE-SPA. *Getting Started with single-spa*. 2024. Disponível em: https://single-spa.js.org/docs/getting-started-overview/. Citado na página 17.

TOTVS. O que é css? conheça benefícios e como funciona. *https://www.totvs.com/blog/developers/o-que-e-css/*, 2020. Acesso em 03 de outubro de 2023. Este é um site que contém informações sobre conceitos de css e seusbenefícios. Citado na página 16.

TOTVS. O que É react: Para que serve, como funciona e caracterÍsticas. *https://kenzie.com.br/blog/react/*, 2021. Acesso em 03 de outubro de 2023. Este é um site que contém informações sobre conceitos de react e suas caracteristicas. Citado na página 11.

VELOSO, A. P. et al. Verdanizze: site de redirecionamento. 107, 2022. Citado na página 16.

WANG, D. et al. A novel application of educational management information system based on micro frontends. *Procedia Computer Science*, Elsevier, v. 176, p. 1567–1576, 2020. Citado na página 19.

WAZLAWICK, R. S. *Metodologia de Pesquisa para Ciência da Computaçâo*. 2021. Acesso em: 25 de junho de 2024. Disponível em: C:dePesquisaparaCiÃłnciadaComputaÃğÃčo.pdf. Citado 2 vezes nas páginas 12 e 21.

YANG, C.; LIU, C.; SU, Z. Research and application of micro frontends. In: IOP PUBLISHING. *IOP conference series: materials science and engineering*. [S.l.], 2019. v. 490, p. 062082. Citado na página 20.